

Graph Neural Networks at a Fraction

Rucha Bhattachandra Joshi^{1*}, Sagar Prakash Barad²³, Nidhi Tiwari⁴, and
Subhankar Mishra²³

¹ The Cyprus Institute, Nicosia, Cyprus

² National Institute of Science Education and Research, Bhubaneswar, India

³ Homi Bhabha National Institute, Mumbai, India

⁴ Microsoft Ltd., India

r.joshi@cyi.ac.cy, sagar.barad@niser.ac.in, nidhitiwari@microsoft.com,
smishra@niser.ac.in

Abstract. Graph Neural Networks (GNNs) have emerged as powerful tools for learning representations of graph-structured data. In addition to real-valued GNNs, quaternion GNNs also perform well on tasks on graph-structured data. With the aim of reducing the energy footprint, we reduce the model size while maintaining accuracy comparable to that of the original-sized GNNs. This paper introduces Quaternion Message Passing Neural Networks (QMPNNs), a framework that leverages quaternion space to compute node representations. Our approach offers a generalizable method for incorporating quaternion representations into GNN architectures at one-fourth of the original parameter count. Furthermore, we present a novel perspective on Graph Lottery Tickets, redefining their applicability within the context of GNNs and QMPNNs. We specifically aim to find the initialization lottery from the subnetwork of the GNNs that can achieve comparable performance to the original GNN upon training. Thereby reducing the trainable model parameters even further. To validate the effectiveness of our proposed QMPNN framework and LTH for both GNNs and QMPNNs, we evaluate their performance on real-world datasets across three fundamental graph-based tasks: node classification, link prediction, and graph classification. Our code is available at project’s GitHub repository.

Keywords: Graph Neural Networks · Pruning · Quaternion Graph Neural Networks · Lottery Ticket Hypothesis.

1 Introduction

Quaternions, a hypercomplex number system, offer several advantages over traditional real and complex numbers, making them well-suited for various tasks in deep learning. Incorporating quaternions into deep neural networks holds significant promise for enhancing their expressive power and versatility. When used in a quaternion setting, Quaternion deep neural networks perform better than

* This research was done while R. Joshi was a student at NISER, Bhubaneswar¹ and HBNI, Mumbai².

their natural counterparts [9]. The added advantage of the quaternions is that they reduce the degree of freedom of parameters by one-fourth, which effectively means that only one-fourth of the parameters must be tuned. The inherent multidimensional nature of quaternions, coupled with their capacity to diminish parameter counts significantly, suggests that hyper-complex numbers are more suitable than real numbers for crafting more efficient models in multidimensional spaces with lesser trainable parameter counts.

Most GNNs learn the embeddings in the Euclidean space. A recent approach [8] proposes learning graph embeddings in Quaternion space. However, it is the quaternion version of one model GCN [5], and it cannot be generalized further. The GNN variants exploit different interesting properties of the graphs to learn the representations efficiently. A quaternion framework should be able to maintain these characteristics of the GNN model and embrace them in the quaternion space. With this realization, in this work, we present a generalizable quaternion framework that can adapt to any GNN variant.

Additionally, unlike in the case of the non-GNN deep neural networks [4,6,7], the existing QGNNs do not leverage the quaternion space to reduce the number of parameters. In these non-GNN methods, the input features are usually divided in such a way that overall the four components of the quaternion vector can aggregate them and make a meaningful transformation out of it. The previous work [8], however, uses each quaternion component to work with all the features. This adds four times the computational overhead, making the model expensive in terms of training and inference time. Additionally, it makes the outputs very sensitive to even a tiny change in the input, as this type of quaternion GNN amplifies the tiny change. To this end, we restrict our quaternion model from quadrupling the number of parameters, thereby reducing the trainable parameter count to one-fourth of that of real, and this also prohibits the output from being extremely sensitive to the tiny change in the input features.

In this work, our goal is to attain Quaternion Graph Neural Networks that are expressive and have fewer parameters. To do this, we *first* propose a generic framework to get the Quaternion model corresponding to any GNN model with real representations. *Secondly*, we present a framework to sparsify the Quaternion GNNs so that we reduce the number of parameters by finding a lottery ticket. We generalize the Lottery Ticket Hypothesis [2] to Quaternion Graph Neural Networks. For graphs, the existing unified graph lottery ticket, which prunes the input graph as well as the GNN to obtain the winning graph lottery ticket, has some drawbacks. Primarily, it is not feasible to prune the input graph along with the GNNs in graph-level tasks. UGS lacks effectiveness for transfer learning purposes because of its dependency on the input graph. We tackle this by redefining the lottery ticket for graphs. Figure 1 presents an overview of our approach.

To summarize, this paper makes contributions as follows: (1) Quaternion Message Passing Neural Networks (QMPNNs): We present a generalizable framework for computing quaternion space representations using any GNN, reducing trainable parameters by 75%. (2) We redefine the graph lottery tickets from a

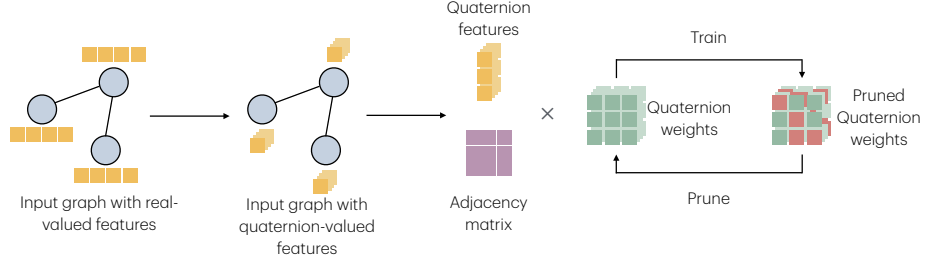


Fig. 1: Pruning a quaternion message passing neural network. Graph’s features are first transformed into quaternion features. We train QMPNN, which has quaternion weights, for the given task. Furthermore, to find our proposed winning lottery ticket, we prune and train the QMPNN until we get a model with pruned quaternion weights that gives a comparable accuracy. The pruned quaternion network can be trained with only a small fraction of the parameters in the real GNN.

different perspective for GNNs and QMPNNs. (3) To our knowledge, we are the first to empirically demonstrate the existence of graph lottery tickets in QMPNNs. (4) We provide the performance evaluation of the proposed LTH on both GNNs and QMPNNs on real-world datasets for three graph-based tasks - node classification, link prediction, and graph classification.

2 Background

2.1 Notations and Definitions

We use the following notations in this paper. A graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ has a set of vertices \mathcal{V} and a set of edges \mathcal{E} . The feature vector corresponding to a node $v \in \mathcal{V}$ is given as \mathbf{x}_v in \mathbb{R}^F . The feature matrix for the graph is given as $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times F}$. The edge $e_{ij} = (v_i, v_j) \in \mathcal{E}$ if and only if there is an edge between two nodes v_i and v_j . In the adjacency matrix representation $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ of the graph, the $\mathbf{A}_{ij} = 1$ if $e_{ij} \in \mathcal{E}$, and 0 otherwise.

The intermediate representations in layer l corresponding to a node v are given by $h_v^{(l)}$. Similarly, we denote representations in layer l for node v by $h_v^{(l),Q}$, using the additional superscript Q to represent quaternion.

2.2 Quaternions

The set of quaternions is denoted by \mathbb{H} . A quaternion q is denoted as $q = q_r + q_i i + q_j j + q_k k$, where $q_r, q_i, q_j, q_k \in \mathbb{R}$ and i, j, k are imaginary units. They follow the relation $i^2 = j^2 = k^2 = ijk = -1$.

Addition: The quaternion addition of two quaternions q and p is component-wise: $(q_r + q_i i + q_j j + q_k k) + (p_r + p_i i + p_j j + p_k k) = (q_r + p_r) + (q_i + p_i) i + (q_j + p_j) j + (q_k + p_k) k$.

Multiplication: Quaternions can be multiplied by each other. The multiplication is associative, i.e., $(pq)r = p(qr)$ for $p, q, r \in \mathbb{H}$. It is also distributive, i.e., $(p+q)r = pr + qr$. However, the multiplication is not commutative, i.e., $pq \neq qp$. When multiplied with the scalar λ , it gives $\lambda q = \lambda q_r + \lambda q_i i + \lambda q_j j + \lambda q_k k$. Multiplication of two quaternions q and p is defined by the Hamilton product $q \otimes p$. This is given in the matrix form as follows:

$$q \otimes p = \begin{bmatrix} 1 \\ i \\ j \\ k \end{bmatrix}^\top \begin{bmatrix} q_r & -q_i & -q_j & -q_k \\ q_i & q_r & -q_k & q_j \\ q_j & q_k & q_r & -q_i \\ q_k & -q_j & q_i & q_r \end{bmatrix} \begin{bmatrix} p_r \\ p_i \\ p_j \\ p_k \end{bmatrix} \quad (1)$$

Conjugation: $\bar{q} = q_r - q_i i - q_j j - q_k k$ is the conjugation of the quaternion q given above.

Norm: The norm of quaternion q is given as $\|q\| = \sqrt{q_r^2 + q_i^2 + q_j^2 + q_k^2}$.

3 Quaternion Message Passing Neural Networks

Graph Neural Networks (GNNs) analyze graph-structured data by updating node representations through message passing. At each layer, messages from neighboring nodes are aggregated and used to update node features. The process is formalized as:

$$m_{uv} = \text{MESSAGE}(h_u^{(l)}, h_v^{(l)}, e_{uv}) \quad (2)$$

$$h'_u = \text{AGGREGATE}(m_{uv}), \forall v \in \mathcal{N}(u) \quad (3)$$

$$h^{(l+1)} = \text{UPDATE}(h_u^{(l)}, h'_u) \quad (4)$$

Here, MESSAGE computes messages based on node features $h_u^{(l)}, h_v^{(l)}$ and edge features e_{uv} ; AGGREGATE collects messages from neighbors $\mathcal{N}(u)$; and UPDATE refines the node representation. These functions, often learnable and differentiable, enable GNNs to be trained via backpropagation.

Quaternion Graph Neural Networks (QGNNs) refer to neural networks that utilize quaternion weights to perform computations within the quaternion vector space. Notably, QGNN, as developed by [8] is a quaternion variant of the Graph Convolutional Network (GCN) layer only. This is the first limitation of QGNN, as it cannot be generalized to other GNN variants. The second limitation of QGNN is by construction, number of trainable parameters remain the same as that of real-valued GCN. While leveraging the quaternion space to generate expressive representations for nodes, we also seek to take advantage of the quaternions by reducing the number of trainable parameters. To this end, we propose a generalized framework to give the quaternion equivalent of any graph neural network method.

In the quaternion variant of any GNNs, in the equations 2, 3, 4, we ensure that the MESSAGE, AGGREGATE and UPDATE functions follow the quaternion operations as given in section 2.2. We describe this in more detail below:

To begin, we have the features represented using the quaternions. F is the number of features corresponding to every node. To use these features with quaternions, we make sure that F is divisible by four so that we can associate $F/4$ real-valued features with a real and three imaginary components of quaternion. Consequently, the feature vector $h_v^{(l),Q}$ of node v in quaternion form is represented as $h_v^{(l),Q} = h_r^{(l)} + h_i^{(l)}\mathbf{i} + h_j^{(l)}\mathbf{j} + h_k^{(l)}\mathbf{k}$.

In every layer l of GNN, the learnable functions have associated parameters, which we represent using $W^{(l),Q}$. This weight matrix $W^{(l),Q}$ has consists of four real-valued components $W_r^{(l)}, W_i^{(l)}, W_j^{(l)}, W_k^{(l)}$ of the quaternion weight matrix. Specifically, $W^{(l),Q} = W_r^{(l)} + W_i^{(l)}\mathbf{i} + W_j^{(l)}\mathbf{j} + W_k^{(l)}\mathbf{k}$.

The weights are multiplied with feature maps using the quaternion multiplication rule given in equation 1. Following this, it is done as follows:

$$W^{(l+1),Q} = W^{(l),Q} \otimes h_v^{(l),Q} \quad (5)$$

$$\begin{bmatrix} \mathcal{R}(W^{(l),Q}) \\ \mathcal{I}(W^{(l),Q}) \\ \mathcal{J}(W^{(l),Q}) \\ \mathcal{K}(W^{(l),Q}) \end{bmatrix} = \begin{bmatrix} W_r^{(l)} & -W_i^{(l)} & -W_j^{(l)} & -W_k^{(l)} \\ W_i^{(l)} & W_r^{(l)} & -W_k^{(l)} & W_j^{(l)} \\ W_j^{(l)} & W_k^{(l)} & W_r^{(l)} & -W_i^{(l)} \\ W_k^{(l)} & -W_j^{(l)} & W_i^{(l)} & W_r^{(l)} \end{bmatrix} \times \begin{bmatrix} h_r^{(l)} \\ h_i^{(l)} \\ h_j^{(l)} \\ h_k^{(l)} \end{bmatrix} \quad (6)$$

where $\mathcal{R}, \mathcal{I}, \mathcal{J}, \mathcal{K}$ denote the four imaginary components of the product of weights with the node representations.

Due to quaternion weights being multiplied with quaternion features, the degrees of freedom are reduced by one-fourth. The product in equation 5 combines components (r, i, j, k) , capturing complex interdependencies. Each quaternion weight, as shown in equation 6, encodes spatial correlations across features, resulting in more expressive networks compared to real-valued counterparts [11]. Additionally, the trainable parameters are reduced to one-fourth of those in an equivalent GNN with real weights, aiding in model size reduction.

3.1 Differentiability

For the QMPNNs to learn, the quaternion versions of learnable functions from MESSAGE, AGGREGATE, and UPDATE should be differentiable. For our framework, the differentiability follows from the generalized complex chain rule for a real-valued loss function, which is provided in much detail in Deep Complex Networks [11] and Deep Quaternion Networks [3].

3.2 Computational Complexity

Four feature values are clubbed together to form a single quaternion in QMPNNs, hence $F/4$ quaternions are necessary to transform the feature vector. The transformed feature space is of dimension F' . While using quaternion weights, these

are considered as the combination of $F'/4$ quaternions. The number of parameters of the quaternion weight matrix, hence, is $F/4 \times F'/4$, with each of them having four quaternion components. Since the components $W_r^{(l)}, W_i^{(l)}, W_j^{(l)}$, and $W_k^{(l)}$ of the quaternion weight $W^{(l),Q}$ are shared during the Hamilton product, the degree of freedom is reduced by one-fourth of that of the real weight matrix. For quaternion matrix multiplications has same time complexity as that of the real ones, we reduce the number of trainable parameters without compromising on the time complexity due to use of quaternion parameters.

4 Lottery Ticket Hypothesis on QMPNNs

The lottery ticket hypothesis (LTH) [2] states that a randomly initialized, dense neural network contains a sub-network that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.

Different from the lottery tickets defined previously [1,12,13,14] - that consider a subgraph of a graph with or without the subnetwork of the GNN as a ticket, we define the LTH for Graph Neural Networks and Quaternion GNNs. In a graph neural network, $f(x; W)$, with initial parameters $W^{(0)}$, using Stochastic Gradient Descent (SGD), we train on the training set, minimizes to a minimum validation loss l in j number of iterations achieving accuracy a . The lottery ticket hypothesis is that there exists m when optimizing with SGD on the same training set, f attains the test accuracy $a' \geq a$ with a validation loss $l' \leq l$ in $j' \leq j$ iterations, such that $\|m\| \ll \|W\|$, thereby reducing the number of parameters. The sub-network with parameters $m \odot W^Q$ is the winning lottery ticket, with the mask m . Algorithm 1 sparsifies the Graph Neural Network which is required to find the winning lottery ticket. The iterative algorithm that finds it is described in algorithm 2.

Algorithm 1 GNN Sparsification

Require: Input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, GNN's initialization $W^{(0)}$, initial mask $m^0 = 1 \in \mathbb{R}^{\|W\|}$, step size η, λ

Ensure: Sparsified weight mask m

- 1: **for** $i \leftarrow 0$ to $N - 1$ **do**
 - 2: Forward $f(\cdot, m^i \odot W)$ with $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ to compute the loss $\mathcal{L}_{\text{task}}$
 - 3: Back-propagate to update $W^{(i+1)} \leftarrow W^{(i)} - \eta \nabla_W \mathcal{L}_{\text{task}}$
 - 4: Update $m^{i+1} \leftarrow m^i - \lambda \nabla_{m^i} \mathcal{L}_{\text{task}}$
 - 5: **end for**
 - 6: Set $p = 20\%$ of the lowest magnitude values in m^N to 0 and others to 1, get mask m
-

We primarily differ from the definition of GLT in the previous works, as we observe that this is not suitable for graph-level tasks [10,8]. Popular methods

such as UGS [1], AdaGLT [14] consider only the node-classification and the link prediction task. In graph-level tasks, sparsifying the input graph does not have any significance. For graph-level tasks, the structure and the features in the entire graph are considered, unlike in the case of node- or edge-level tasks where only the nodes in the local neighborhood and their features are considered by a graph neural network. Also, in the case of transfer learning, where the pruned sub-network is to be further finetuned for a different but related task, the dependency of the winning ticket on the input graph restricts us from fine-tuning it.

4.1 Quaternion Graph Lottery Tickets

Similar to finding a winning lottery ticket in GNNs, given a QMPNN $f(\cdot, W^Q)$ and a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the subnetwork, i.e., winning lottery ticket, of the QMPNN is defined as $f(\cdot, m \odot W^Q)$, where m is the binary mask on parameters. To find the winning lottery ticket in a QMPNN, we use the algorithms 1 and 2 with the quaternion-values model weights and input features.

Algorithm 2 Iterative algorithm to find Graph Lottery Ticket

Require: Input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, GNN’s initialization $W^{(0)}$, initial mask $m^0 = 1 \in \mathbb{R}^{\|W\|}$, sparsity level s

Ensure: Lottery Ticket $f((\mathcal{V}, \mathcal{E}, \mathbf{X}), m \odot W^0)$

- 1: **while** $1 - \frac{\|m\|}{\|W\|}$ **do**
 - 2: Sparsify GNN $f(\cdot, m \odot W^0)$ using algorithm 1
 - 3: Update m according to step 2
 - 4: Reset GNN’s weight to W^0
 - 5: **end while**
-

5 Experiments

The effectiveness of QMPNNs for different GNN variants is validated with extensive experimentation on different real-world standard datasets. We also verify the existence of the GLTs as per our definition. We evaluated their performance on three tasks, node classification, link prediction, and graph classification.

5.1 Datasets

Tables 1 and 2 summarize the datasets and their statistics. For quaternion models, dataset features and classes were adjusted to be divisible by 4 by padding features with the average value and adding dummy classes. For instance, Cora’s feature size was padded from 1,433 to 1,436, and the number of classes was increased from 7 to 8.

Table 1: Node classification and link prediction datasets statistics

Dataset	<i>#Nodes</i>	<i>#Edges</i>	<i>#Features</i>	<i>#Classes</i>	<i>Metric</i>
Cora	2,708	5,429	1,433	7	Accuracy, ROC-AUC
Citeseer	3,327	4,732	3,703	6	Accuracy, ROC-AUC
PubMed	19,717	44,338	500	3	Accuracy, ROC-AUC
ogbn-arxiv	169,343	1,166,243	128	40	Accuracy, ROC-AUC
ogbl-collab	235,868	2,358,104	128	0	Hits@50

Table 2: Graph classification datasets statistics

Dataset	<i>#Graphs</i>	<i>Avg. #Edges</i>	<i>#Classes</i>	<i>Metric</i>
MUTAG	188	744	2	Accuracy
ENZYMES	600	1,686	6	Accuracy
PROTEINS	1,113	3,666	2	Accuracy
ogbg-molhiv	41,127	40	2	Accuracy

5.2 Experimental Setup

- **GNN Architecture:** Two-layer GCN, GAT, and GraphSAGE models with 128 hidden units were used for node classification and link prediction on Cora, Citeseer, PubMed, ogbn-arxiv, and ogbn-collab. For graph classification on MUTAG, PROTEINS, ENZYMES, and ogbg-molhiv, three-layer models with 128 hidden units per layer were employed.
- **Training and Hyperparameters:** Node classification datasets followed an 80-10-10 split, while link prediction and graph classification used an 85-5-10 split. Models were trained with a learning rate of 0.01, weight decay of 5×10^{-4} , dropout rate of 0.6, and the Adam optimizer, for up to 1000 epochs with early stopping after 200 epochs of no improvement. A prune fraction of 0.3 was applied.
- **Evaluation and Infrastructure:** Evaluation metrics are provided in Tables 1 and 2. Experiments were conducted on NVIDIA RTX 3090 GPUs with 24GB VRAM.

5.3 Training and Inference Details

Our evaluation metrics, detailed in Tables 1 and 2 and following [5], include both accuracy and ROC-AUC. Accuracy measures the proportion of correctly classified instances among the total instances, providing a straightforward metric for model performance. The ROC-AUC (Receiver Operating Characteristic - Area Under the Curve) score, representing the degree of separability, is particularly valuable for assessing performance on imbalanced datasets, such as in link prediction tasks with a substantial class imbalance between positive and negative edges. Summarizing the results across these diverse tasks, tables 3 , 4,

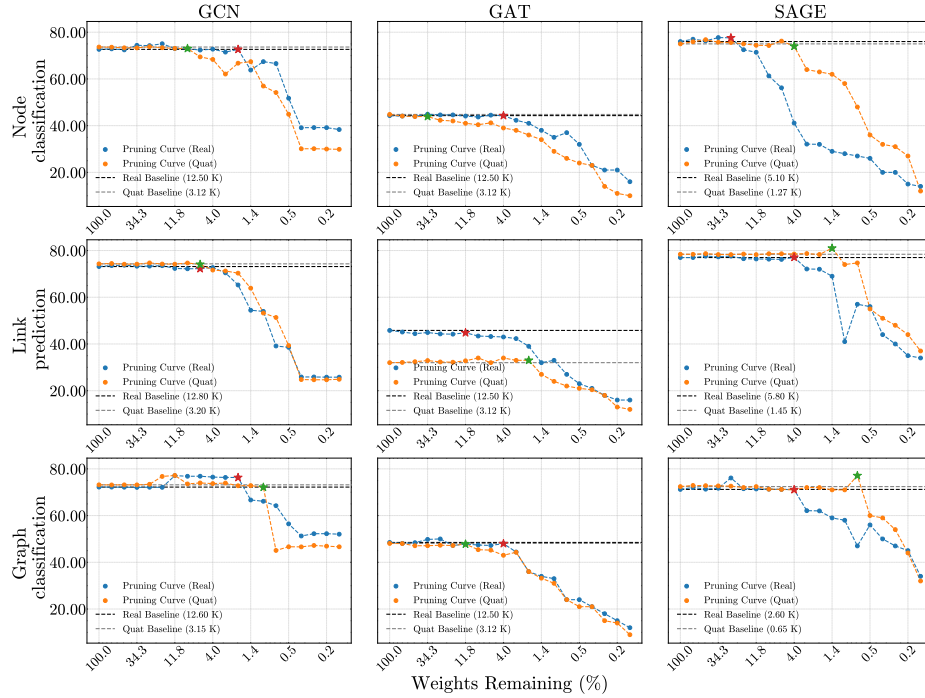


Fig. 2: **Performance After Pruning:** The plots show GCN, GAT, and GraphSAGE performance on OGBN-ARXIV (node classification), OGBL-COLLAB (link prediction), and OGBG-MOHLIV (graph classification) at a pruning weight fraction of 0.3. **GLTs** are marked by red (★) and green (★) stars, indicating comparable performance despite sparsity, while dashed lines represent pre-pruning baselines. The plot sections correspond to node classification (top), link prediction (middle), and graph classification (bottom).

and 5 present the inference outcomes for all models, facilitating comprehensive analysis.

5.4 Results

GCN, GAT, and GraphSAGE results on Cora, Citesser, PubMed, and ogbn-arxiv for node classification are collected in table 3 and 4 shows results for link prediction on Cora, Citesser, PubMed, and ogbl-collab. The results pertaining to graph classification tasks on MUTAG, PROTEINS, ENZYMES, and ogbg-molhiv datasets are presented in table 5. These tables display the accuracy and parameter count (Params) in millions for each model and dataset, with better-performing models highlighted in bold for easy identification. Pruning results for the same models on ogbn-arxiv for node classification are shown in the top section of figure 2. The middle section of figure 2 presents results for link prediction on

the ogbl-collab dataset. The bottom section of figure 2 illustrates the results for graph classification tasks on the ogbg-molhiv dataset. An extensive list of results is available at project’s GitHub repository.

Table 3: **Node classification** results for GCN, GAT, and GraphSAGE models on semi-supervised graph datasets, including Cora, Citeseer, PubMed, and OGBN-ARXIV. Accuracy values are reported with their standard deviations, along with the corresponding GFLOPs (denoted as GF) and parameter counts (in K units, denoted as GF).

Model	Cora			Citeseer			PubMed			OGBN-Arxiv		
	Accuracy	Par (K)	GF	Accuracy	Par (K)	GF	Accuracy	Par (K)	GF	Accuracy	Par (K)	GF
GCN	85.2 \pm 1.3	140.5	0.33	73.5 \pm 1.8	125.7	0.29	79.1 \pm 1.2	131.9	0.31	72.8 \pm 1.5	145.1	0.42
QGCN	84.1 \pm 1.5	42.1	0.11	72.6 \pm 1.3	39.2	0.09	78.4 \pm 1.4	41.7	0.10	72.3 \pm 1.6	43.5	0.12
GAT	87.3 \pm 1.0	143.2	0.37	75.4 \pm 1.9	130.8	0.34	80.7 \pm 1.6	136.3	0.36	72.5 \pm 1.2	149.5	0.45
QGAT	86.4 \pm 1.7	43.7	0.14	74.8 \pm 1.1	41.6	0.12	79.8 \pm 1.5	42.9	0.13	72.9 \pm 1.8	44.5	0.15
SAGE	85.5 \pm 1.2	72.6	0.23	74.9 \pm 1.4	66.1	0.19	80.2 \pm 1.9	68.7	0.20	73.0 \pm 1.0	75.3	0.26
QSAGE	84.9 \pm 0.8	22.1	0.08	74.3 \pm 1.5	20.5	0.07	80.5 \pm 1.3	21.3	0.08	72.8 \pm 1.4	23.6	0.10

The results indicate that quaternion models (QGCN, QGAT, and QSAGE) perform equally well or better than their real counterparts (GCN, GAT, and GraphSAGE) under the same hyperparameters and training parameters. For instance, QGCN shows slightly improved accuracy on the PubMed dataset, and QGAT performs nearly as well as GAT on Cora and Citeseer for node classification tasks. Moreover, quaternion models consistently outperform their real counterparts in other tasks, including link prediction and graph classification, demonstrating superior performance in almost all cases. Furthermore, it is consistently shown that even when the real models perform better than QMPNNs, they are mostly ahead by only marginal values.

Table 4: **Link prediction** on various datasets including CORA, CITESEER, PUBMED, and OBGL-COLLAB. ROC-AUC and Hits@50 values are reported with their standard deviations, along with the corresponding GFLOPs (denoted as GF) and parameter counts (in K units, denoted as GF).

Model Name	CORA			CITESEER			PUBMED			OGBL-COLLAB		
	ROC-AUC	Par (K)	GF	ROC-AUC	Par (K)	GF	ROC-AUC	Par (K)	GF	Hits@50	Par (K)	GF
GCN	79.27 \pm 0.47	96.0	0.23	78.53 \pm 0.45	241.4	0.58	87.60 \pm 0.043	36.3	0.09	44.02 \pm 1.57	12.5	0.03
QGCN	83.55 \pm 0.03	27.2	0.07	78.36 \pm 4.13	63.6	0.15	87.66 \pm 0.32	12.3	0.03	45.80 \pm 1.35	6.4	0.02
GAT	93.89 \pm 0.06	96.3	0.24	85.56 \pm 0.38	241.6	0.60	89.32 \pm 0.38	36.6	0.09	45.51 \pm 1.61	12.8	0.03
QGAT	94.99 \pm 0.19	27.5	0.07	86.59 \pm 0.10	63.8	0.16	84.04 \pm 0.54	12.6	0.03	32.03 \pm 1.92	65.6	0.16
SAGE	89.00 \pm 0.19	48.1	0.12	81.65 \pm 0.13	120.5	0.30	83.36 \pm 0.0189	16.1	0.04	48.50 \pm 0.88	6.4	0.02
QSAGE	89.35 \pm 0.26	13.6	0.03	81.93 \pm 0.04	31.8	0.08	79.05 \pm 0.32	6.1	0.02	48.07 \pm 0.56	3.2	0.01

The results presented in the tables comparing real and quaternion models, and are trained and evaluated using five different seeds. For each data point,

we calculated the mean and standard deviation based on these evaluations. The better-performing models are determined by performing paired t-tests. We chose the model with the better performance when the difference between p -values of the models was found to be statistically significant.

Table 5: **Graph Classification** on various datasets including MUTAG, PROTEINS, ENZYMES, and OGBG-MOLHIV. Accuracy values are reported with their standard deviations, along with the corresponding GFLOPs (denoted as GF) and parameter counts (in K units, denoted as GF).

Model Name	MUTAG			PROTEINS			ENZYMES			OGBG-MOLHIV		
	Accuracy	Par (K)	GF	Accuracy	Par (K)	GF	Accuracy	Par (K)	GF	Accuracy	Par (K)	GF
GCN	83.54 \pm 0.40	4.8	0.01	73.57 \pm 0.45	9.2	0.02	72.96 \pm 0.15	5.6	0.01	76.24 \pm 0.89	5.1	0.01
QGCN	88.61 \pm 0.66	4.4	0.01	71.55 \pm 0.42	5.5	0.01	71.92 \pm 1.16	4.6	0.01	75.85 \pm 0.93	4.5	0.01
GAT	85.26 \pm 0.02	5.4	0.01	72.68 \pm 0.93	9.4	0.02	74.61 \pm 0.05	5.8	0.01	76.24 \pm 0.32	5.8	0.01
QGAT	87.25 \pm 0.18	4.7	0.01	73.22 \pm 0.02	5.8	0.01	73.35 \pm 0.02	4.9	0.01	78.16 \pm 0.78	4.8	0.01
SAGE	82.56 \pm 2.67	2.5	0.01	72.62 \pm 0.22	4.7	0.01	79.65 \pm 3.06	2.8	0.01	71.77 \pm 0.97	2.6	0.01
QSAGE	84.26 \pm 0.22	2.2	0.01	73.08 \pm 1.36	2.7	0.01	78.59 \pm 0.73	2.3	0.01	72.98 \pm 0.12	2.2	0.01

It is worth noting that the performance of real-valued GNN models, on datasets with and without the dataset augmentation mentioned in the section 5.1 is the same. Hence, it serves as a fair baseline to compare the quaternion-valued GNN performance with the real-valued GNN performance on the augmented dataset. We provide the code in the repository linked in the paper.

5.5 Key Findings

We summarize the effectiveness of magnitude pruning and quaternion-based models in GNNs:

1. **QMPNNs operate at 1/4th parameters of real-valued models:** QMPNNs achieve comparable or better performance than real-valued models with just 1/4th of the parameters, showcasing their efficiency and expressiveness, as shown in Tables 3, 4, and 5.
2. **GLTs exist at 1/5th or smaller of original size:** Magnitude pruning identifies lottery tickets at 1–20% of the original model size without performance loss across QGCN, QGAT, and QGraphSAGE tasks, as observed in Tables 3 and 4.
3. **GAT and GraphSAGE sparsify well; Cora is pruning-sensitive:** GATs and GraphSAGE produce sparse GLTs due to attention and sampling techniques, while Cora exhibits significant sensitivity to pruning, with performance dropping at half model size, as detailed in the comprehensive experiments available on the project’s GitHub repository.
4. **Less sparsity in graph classification GLTs:** Graph classification tasks require less sparse GLTs to capture complex global features, with smaller parameter counts due to the datasets’ limited graph size, as shown in Table 5 and Figure 2.

5. **Large graphs maintain performance:** QMPNNs scale well to large graphs, as seen with ogbn-arxiv, ogbn-collab, and ogbn-molhiv datasets in Tables 3, 4, and 5, delivering comparable or better performance relative to trainable parameters.

6 Conclusion and Future Work

This paper introduces Quaternion Message Passing Neural Networks (QMPNNs) as a versatile framework for graph representation learning, leveraging quaternion representations to capture intricate relationships in graph-structured data. The framework generalizes easily to existing GNN architectures with minimal adjustments, enhancing flexibility and performance across tasks like node classification, link prediction, and graph classification. By redefining graph lottery tickets, we identified key subnetworks that enable efficient training and inference, demonstrating the scalability and effectiveness of QMPNNs on real-world datasets. Future work could explore dynamic graphs, hybrid modalities, and improving interpretability to further expand QMPNN capabilities.

References

1. Chen, T., Sui, Y., Chen, X., Zhang, A., Wang, Z.: A unified lottery ticket hypothesis for graph neural networks. In: International conference on machine learning. pp. 1695–1706. PMLR (2021)
2. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: International Conference on Learning Representations (2018)
3. Gaudet, C.J., Maida, A.S.: Deep quaternion networks. In: 2018 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE (2018)
4. Iqbal, S.M., Mishra, S.: Neural networks at a fraction with pruned quaternions. In: Proceedings of the 6th Joint International Conference on Data Science & Management of Data (10th ACM IKDD CODS and 28th COMAD). pp. 19–27 (2023)
5. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (2016)
6. Mukhopadhyay, A., Adhilsha, A., Mishra, S.: Large neural networks at a fraction. In: Northern Lights Deep Learning Conference. pp. 165–173. PMLR (2024)
7. Mukhopadhyay, A., Joshi, R.B., Tiwari, N., Mishra, S.: Transformers at a fraction. In: Northern Lights Deep Learning Conference 2025
8. Nguyen, T.D., Phung, D., et al.: Quaternion graph neural networks. In: Asian conference on machine learning. pp. 236–251. PMLR (2021)
9. Parcollet, T., Morchid, M., Linarès, G.: A survey of quaternion neural networks. Artificial Intelligence Review **53**(4), 2957–2982 (2020)
10. Sun, X., Cheng, H., Li, J., Liu, B., Guan, J.: All in one: Multi-task prompting for graph neural networks. In: Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. p. 2120–2131. KDD '23, Association for Computing Machinery, New York, NY, USA (2023). <https://doi.org/10.1145/3580305.3599256>, <https://doi.org/10.1145/3580305.3599256>

11. Trabelsi, C., Bilaniuk, O., Zhang, Y., Serdyuk, D., Subramanian, S., Santos, J.F., Mehri, S., Rostamzadeh, N., Bengio, Y., Pal, C.J.: Deep complex networks. In: International Conference on Learning Representations (2018), <https://openreview.net/forum?id=H1T2hmZAb>
12. Tsitsulin, A., Perozzi, B.: The graph lottery ticket hypothesis: Finding sparse, informative graph structure. In: NeurIPS 2023 Workshop: New Frontiers in Graph Learning (2023), <https://openreview.net/forum?id=shKEocsDbB>
13. Wang, K., Liang, Y., Wang, P., Wang, X., Gu, P., Fang, J., Wang, Y.: Searching lottery tickets in graph neural networks: A dual perspective. In: The Eleventh International Conference on Learning Representations (2023), <https://openreview.net/forum?id=Dvs-a3aymPe>
14. Zhang, G., Wang, K., Huang, W., Yue, Y., Wang, Y., Zimmermann, R., Zhou, A., Cheng, D., Zeng, J., Liang, Y.: Graph lottery ticket automated. In: The Twelfth International Conference on Learning Representations (2024), <https://openreview.net/forum?id=nmBjBZoySX>